L Number	Hits	Search Text	DB	Time stamp
-	44	(listen$3 adj5 queue$3) and (connect$4 or session$3)	USPAT	2001/10/19 09:36
-	44	((listen$3 adj5 queue$3) and (connect$4 or session$3)) and tim$3	USPAT	2001/10/19 09:37
-	44	((listen$3 adj5 queue$3) and (connect$4 or session$3)) and tim$5	USPAT	2001/10/19 09:37
-	13	((listen$3 adj5 queue$3) and (connect$4 or session$3)) and (tim$5 adj3 out)	USPAT	2001/10/19 09:38
-	46	listen$3 adj5 queue$3	USPAT	2001/10/19 09:54

US-PAT-NO: 6125401
DOCUMENT-IDENTIFIER: US 6125401 A
TITLE: Server detection of client process termination
DATE-ISSUED: September 26, 2000
INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Huras; Matthew A.	Ajax	N/A	N/A	CAX
Vincent; Tim J.	Toronto	N/A	N/A	CAX

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY	N/A	N/A	02

APPL-NO: 8/ 623185
DATE FILED: March 28, 1996
FOREIGN-APPL-PRIORITY-DATA:
FOREIGN-PRIORITY:
FOREIGN-PRIORITY-APPL-NO: CA 2146170
FOREIGN-PRIORITY-APPL-DATE: April 3, 1995
INT-CL: [7] G06F015/163,G06F009/00 ,G06F009/46
US-CL-ISSUED: 709/300,711/147
US-CL-CURRENT: 709/310,711/147
FIELD-OF-SEARCH: 395/726;395/200.33 ;395/200.44 ;395/651

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5313638	May 1994	Ogle et al.	395/726
5394551	February 1995	Holt et al.	395/726
5553242	September 1996	Russell et al.	395/200.57
5623670	April 1997	Bohannon et al.	395/726
5652885	July 1997	Reed et al.	395/651

ART-UNIT: 275
PRIMARY-EXAMINER: Banankhah; Majid A.
ASSISTANT-EXAMINER: Caldwell; Pat
ATTY-AGENT-FIRM: Gates & Cooper
ABSTRACT:
A service provider for use in a client-server system which is capable of detecting the abnormal termination of a client process is disclosed. The service provider does not require a dedicated process for polling client processes in order to verify their status. Rather, a semaphore, which is used in conjunction with a shared memory segment for communication between a client process and the service provider, is initialized in such a manner that the operating system will automatically increment the semaphore in the event the client process is terminated. Thus, the semaphore will be incremented either when the client process deliberately increments the semaphore in order to notify the service provider that the client process has written data to a shared memory segment, or the semaphore will be incremented by the operating system in the event the client process terminates. A test flag is established in shared memory in order to differentiate whether the semaphore was incremented by the client process, or by the operating system. The client process will set the flag only when the client process increments the semaphore. Therefore, whenever the semaphore is incremented, the service provider will test the condition of the flag, and terminate resources allocated to the client process if the flag is not set.

29 Claims, 3 Drawing figures

DEPR:
The preferred embodiment of the present invention will now be described with reference to FIG. 1. Box 100 represents a single machine computer system, for

example a mainframe computer system, which includes a CPU, memory, disc storage, etc. This figure illustrates schematically in block diagram form the components of a client-server system incorporating the preferred embodiment of the present invention. The figure also illustrates the interaction of these components for allowing inter-process communication between client processes and server processes running on the computer system. The left hand portion of FIG. 1 shows a terminal 141, and a personal computer 151, each **connected** to the computer system and communicating with client process 140 and client process 150 respectively, with each client process running on the main computer system. For ease of illustration, FIG. 1 only illustrates two client processes, although many more would be running concurrently in a typical client-server system. The middle portion of the figure illustrates the operating system inter-process communication resources established by the service provider for enabling data transfer between client and server processes. These resources, which are all labelled with numbers between 200 and 299, include a shared memory segment, and a set of semaphores for each client process. The right hand portion generally illustrates the service provider components within box 300, with each component labelled with numbers between 300 and 399.

DEPR:
Server listener code 305 represents a portion of the service provider which establishes a server listener process 310, which in turn manages each initial client-server interface. As part of the start up of the service provider, the server listener process 310 establishes two known inter-process communication resources, namely a **Listener Response Queue** (a queue for receiving initial messages from client processes) and a **Listener Response Queue** (a queue for responding to the initial messages from client processes), in order to facilitate initial communication between each new client process and the service provider.

DEPR:
Upon loading the server library code 330, the client process 140 will send a message 145 to the Server **listener process 310 by means of the Listener Response Queue,** in a known manner. This message notifies the server listener process 310 that a new client process has been established.

DEPR:
The server listener process 310 then sends the identification information for the semaphores and shared memory segment to the client process 140 (in a known manner by means of the **listener response queue**) as indicated by arrow 308.

US-PAT-NO: 6163812
DOCUMENT-IDENTIFIER: US 6163812 A
TITLE: Adaptive fast path architecture for commercial operating systems and information server applications
DATE-ISSUED: December 19, 2000
INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gopal; Ajei	Fort Lee	NJ	N/A	N/A
Neves; Richard	Tarrytown	NY	N/A	N/A
Vajracharya; Suvas	Boulder	CO	N/A	N/A

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY	N/A	N/A	02

APPL-NO: 8/ 954710
DATE FILED: October 20, 1997
INT-CL: [7] G06F015/163
US-CL-ISSUED: 709/310
US-CL-CURRENT: 709/310
FIELD-OF-SEARCH: 709/300;709/303 ;709/305 ;709/235 ;709/310 ;709/330 ;710/260 ;370/355

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5210832	May 1993	Maier et al.	712/227
5561799	October 1996	Khalidi et al.	707/200
5724514	March 1998	Arias	709/235
5758168	May 1998	Mealey et al.	710/260
5808911	September 1998	Tucker et al.	709/303
5903559	March 1999	Acharya et al.	370/355

ART-UNIT: 278
PRIMARY-EXAMINER: Coulter; Kenneth R.
ATTY-AGENT-FIRM: Ellenbogen; Wayne L.
ABSTRACT:
A general, event/handler kernel extension system is implemented. Network server extension architecture isolates and exploits the ability to derive responses on the same interrupt the original request was received on using non-paged memory. TCP network server extensions are implemented. A technique is defined for facilitating immediate completion of **connection** requests using pre-allocated **connection** endpoints and describes an approach to recycling these **connection** endpoints. A hybrid HTTP extension implemented partially in user space and partially in kernel space is defined that provides explicit or transparent implementation of the user space component and shared logging between user and kernel space. A technique is defined for prefetching responses to HTTP GET requests using earlier GET responses. Classifying of handler extensions according to latency in deriving a response to a network request is defined. Tight integration with the file system cache is available for sending non-paged responses from the file system cache to the remote client. A complete caching scheme is defined for protocols serving file contents to remote clients.

29 Claims, 8 Drawing figures

ABPL:
A general, event/handler kernel extension system is implemented. Network server extension architecture isolates and exploits the ability to derive responses on the same interrupt the original request was received on using non-paged memory. TCP network server extensions are implemented. A technique

is defined for facilitating immediate completion of **connection** requests using pre-allocated **connection** endpoints and describes an approach to recycling these **connection** endpoints. A hybrid HTTP extension implemented partially in user space and partially in kernel space is defined that provides explicit or transparent implementation of the user space component and shared logging between user and kernel space. A technique is defined for prefetching responses to HTTP GET requests using earlier GET responses. Classifying of handler extensions according to latency in deriving a response to a network request is defined. Tight integration with the file system cache is available for sending non-paged responses from the file system cache to the remote client. A complete caching scheme is defined for protocols serving file contents to remote clients.

DRPR:
FIG. 5 is a block diagram which illustrates **connection** management for TCP/IP.

DEPR:
A port is a number. Remote clients send packets to a server. Remote clients specify the port number in the packets they send to a server. A server application "listens" for these packets. When a packet arrives with the port number the server is listening for, the server establishes a **connection** with the remote client. Sockets are a known means for a server application to use TCP/IP. One socket function is called "listen". One of the parameters for "listen" is the port number. When a server calls listen() with a port number, for example port 80, the server is listening for packets associated with port 80 from a remote client. The use of ports is described, for example, in Stevens, W. R., Unix Network Programming, Prentice Hall, 1990.

DEPR:
Resource management and allocation for events occurring within Network Server Extension Architecture 130 on port 80 use a context pointer as a reference to the resources associated with port 80 and the server extension which registered port 80. Thus, resources such as memory and **connection** end points are allocated and used separately on behalf of network server extension 150 by Network Server Extension Architecture 130. These same handler invocation and extension registration capabilities are found in device driver frameworks on modern operating systems. One such example is NT where a device driver framework referred to as the "port"/"miniport" model is used extensively. In this framework, a "port" driver (not to be confused with a network port) allows multiple device drivers called "miniport" drivers to register themselves. Each time a new registration takes place, a new device context is created. On NT, the device context associates resource usage and management with that particular miniport driver. NT provides two port architectures, the specification of which can be adapted to the needs of this invention. The first is the SCSI port architecture. The second is the NDIS port architecture.

DEPR:
Network server portability architecture 120 is described with reference to FIG. 2. Examples of portability handlers 122 and portability services 126 are given. The portability handlers provided in the network server portability architecture are the TCP **connect** handler 210, TCP receive handler 212, and the TCP disconnect handler 214. TCP **connect** handler 210 is executed when TCP **connections** are established with a remote client. TCP receive handler 212 is executed when TCP packets arrive from a remote client. TCP disconnect handler 214 is executed when TCP disconnect requests arrive from a remote client. Windows NT provides a means to execute such handlers. The Windows NT native kernel environment provides events 112 that map directly to the portability handlers 210, 212, 214. Windows NT defines these kernel events as transport driver interface (TDI). The portability handlers may be defined to work with TDI.

DEPR:
Connection, requests, and messages are now defined. **Connections** are data structures defined by the native kernel environment 110, but allocated and deallocated via AFPA run-time support 134. An example of a **connection** is a TCP **connection** endpoint created on the server for tracking **connections** made on a unique address/port pair.

DEPR:
A request has four components: **connection,** request data, request ID, and send data. A server handling multiple requests over the same **connection** (such as FTP) generates requests with the same **connection**. A server handling one request per **connection** (such as HTTP) has a unique **connection** for each request.

DEPR:
FIG. 5 is an illustration of **connection** management for TCP/IP. The remote client portion of the figure shows the actions occurring from the perspective of the remote client computer. Likewise, the server side shows the actions from the perspective of an exemplary embodiment of the present invention.

DEPR:
AFPA 130 maintains a pool of unused **connection** data structures in **connection** pool 550. As **connections** are established, their data structures are allocated from **connection** pool 550. As disconnections occur, **connection** data structures are returned to **connection** pool 550. This reduces the computation necessary to establish **connections**.

DEPR:
When the remote client issues a **connect** request 510, the server computer receives a TCP/IP SYN packet 520. This triggers a native kernel event that executes **connect** handler 210. **Connect** handler 210 executes **connection** management code within AFPA 134. This may include, for example, **connection** management code 540, 590, and 593. Upon receiving the TCP/IP SYN packet, the server computer allocates a new **connection** 540 from the **connection** pool 550. On Windows NT a **connection** is referred to as a **connection** endpoint On UNIX operating systems, a **connection** is typically referred to as a socket. In a preferred embodiment of the present invention **listen and socket queues** are excluded. When a SYN packet arrives it is immediately allocated a **connection** 540 instead of being queued. Completing a TCP **connection** may be indicated by the arrival of a TCP SYN packet. This may be accomplished by sending a TCP SYN.backslash.ACK packet at the same time an interrupt or received for the TCP SYN packet.

DEPR:
A **connection** is destroyed one of two ways: 1) The remote client closes the **connection** FIN packet 570 and disconnect handler 214. 2) The server initiates a disconnect 593 and 220. In either case, the **connection** is returned to the **connection** pool 590 or 593 for use in establishing a new **connection** 540.

DEPR:
When a receive event occurs 610, a data packet 620 arrives. Receive handler 212 is executed which in turn executes DeriveCachedResponse 332 of AFPA 130. DeriveCachedResponse 332 parses the request, determining if it is complete. To accommodate the newly arrived data, AFPA 130 removes a request data structure 640 from a pool of preallocated request data structures 660. AFPA 130 also preallocates message fragments from the message pool 670. For each new data packet, a new message fragment is allocated from the message fragment pool and placed on the message fragment list 680 which is part of the request data structure 640. DeriveCachedResponse 332 then copies the newly arrived data into the message fragment. When enough data has arrived on the **connection** to constitute a full request, DeriveCachedResponse 332 parses the request and queries the cache for the response.

DEPL:
AFPA 130 implements run-time support 134 for a number of items. These items are **connection** management 310, request management 312, cache management 314, thread management 316, and queue management 318. **Connection** management is discussed with reference to FIG. 5. Request management is discussed with reference to FIG. 6. Cache management is discussed with reference to FIG. 7. Queue management means providing an logical queue abstraction with interfaces and multiprocessor safe modification. Unless otherwise specified, queues in this invention are FIFO (first in first out) and are conventional. Thread management means creating and destroying threads. Thread management also means consuming resources from queues without race conditions. Thread management may be implemented in accordance with conventional techniques.

DEPV:
Event--This term refers to the execution of discrete code in response to control flow in a software system. The code is typically executed as a result of a hardware state, such as a hardware interrupt. Examples of network kernel events are TCP packet arrival, TCP **connection** establishment, and TCP disconnection. Network events are initiated by hardware interrupt to the CPU by a network adapter. These hardware interrupts result in TCP protocol events.

CLPR:
5. The system of claim 4 where completing a TCP **connection** as indicated by the arrival of a TCP SYN packet is accomplished by sending a TCP SYN/ACK packet on a same interrupt or thread of execution as the processing required to receive the said TCP SYN packet.

CLPR:
17. The system of claim 14 wherein said extension architecture is a network extension architecture which reuses TCP/IP **connection** endpoints after these **connection** endpoints are disconnected, wherein said **connection** endpoints are data structures associated with individual TCP/IP **connections** in a TCP/IP implementation native to an operating system.

CLPR:
18. The system of claim 1 wherein the extension architecture is a network server extension architecture, said network server extension architecture comprising one or more of: a **connection** endpoint management means for allocating and garbage collecting data structures associated with a network transport layer **connection** endpoints; a request management logic means for managing request fragments on behalf of said supplied handler routines; a cache management logic for deriving responses from non-paged memory; and a thread management logic means for deriving responses on high latency requests.

CLPV:
a) **connection** establishment (TCP/IP SYN packet arrival);

US-PAT-NO: 6202036
DOCUMENT-IDENTIFIER: US 6202036 B1
TITLE: End-to-end response time measurement for computer programs using starting and ending queues
DATE-ISSUED: March 13, 2001
INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Klein; Paul F.	Thousand Oaks	CA	N/A	N/A
Ammerman, III; Raymond P.	Raleigh	NC	N/A	N/A

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Candle Distributed Solutions, Inc.	El Segundo	CA	N/A	N/A	02

APPL-NO: 9/ 428271
DATE FILED: October 27, 1999
PARENT-CASE:
This application is a Continuation of application Ser. No. 08/899,195, filed Jul. 23, 1997 U.S. Pat. No. 5,991,705, entitled END-TO-END RESPONSE TIME MEASUREMENT FOR COMPUTER PROGRAMS USING STARTING AND ENDING QUEUES, which application is incorporated herein by reference.
INT-CL: [7] G04F010/00,G06F013/00
US-CL-ISSUED: 702/178,702/178 ,702/182 ,709/233
US-CL-CURRENT: 702/178,702/182 ,709/233
FIELD-OF-SEARCH: 702/185;702/186 ;702/176 ;702/178 ;714/26 ;714/38 ;714/20 ;348/180 ;709/233

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4868785	September 1989	Jordan et al.	702/185
5068814	November 1991	Stark et al.	702/185
5483468	January 1996	Chen et al.	702/186
5506955	April 1996	Chen et al.	714/26
5511185	April 1996	Weinbaum et al.	714/38
5519438	May 1996	Elliott et al.	348/180
5553235	September 1996	Chen et al.	714/20

FOREIGN PATENT DOCUMENTS

COUNTRY	FOREIGN-PAT-NO	PUBN-DATE	US-CL
EP	0259224	August 1987	

ART-UNIT: 287
PRIMARY-EXAMINER: Shah; Kamini
ATTY-AGENT-FIRM: Gates & Cooper LLP
ABSTRACT:
An end-to-end response time measurement method monitors the performance of a computer program by measuring the time between related messages that traverse inbound and outbound message queues.

18 Claims, 3 Drawing figures

DEPR:
FIG. 1 illustrates an exemplary hardware environment that could be used to implement the preferred embodiment of the present invention. The exemplary hardware environment may include, inter alia, a client computer 100 and/or a server computer 102 **connected** to the client 100. Both the client 100 and server 102 generally include, inter alia, a processor, random access memory (RAM), read only memory (ROM), a monitor 104, data storage devices, data communications devices, etc. The client 100 and server 102 may also include

data input devices such as a mouse pointing device 106 and a keyboard 108. Of course, those skilled in the art will recognize that any combination of the above components, or any number of different components, peripherals, and other devices, may be used with the client and/or server.

DEPR:
In the preferred embodiment of the present invention, the operating system provides the ability for the monitor program to examine the content of messages on a given message queue. This interface is provided through an Application Program Interface (API) provided by the operating system. To compute an application's end-to-end response time, the monitor program issues the appropriate API call and registers itself as a **listener of all messages in a queue**. Thereafter, any messages that traverse the queue are also presented to the monitor program.

DEPR:
As a message is examined, its process id (pid), thread id (tid), message queue handle (msgq) and **session** id (sessid) are determined through standard API functions provided by the operating system. If the message is one of the above, the list 118 is searched backwards to find an active list 118 entry with the corresponding pid, tid and msgq. An active entry is defined as a list 118 entry that has been initiated but not yet marked closed.